سید محمد طاها طباطبایی – تمرین سری دوم
۹۸۱۲۷۶۲۸۳۸

چکیده:
در بخش اول، به بررسی و تحلیل هیستوگرام و متعادل سازی، تبدیل و اعمال توابعی که شدت روشنایی پیکسل ها را تحت تاثیر قرار می‌دهند، پرداختیم. در بخش دوم، به طور مشخص، به مقایسه تاثیر متعادل سازی کلی بر روی تصویر camera man پرداخته ایم. در پارت آخر، متعادل سازی محلی را روی ۴ نمونه تصویر مختلف و با سایز پنجره های متفاوت اعمال کردیم، تا بتوانیم تاثیر ابعاد پنجره مختلف و همچنین تاثیر متعادل سازی در تصاویر با نرخ جزئیات متفاوت را بررسی کنیم.

۲.۱.۱

توضیح فنی:

برای محاسبه هیستوگرام یک عکس مطابق عکس زیر، ابتدا با کمک لایبرری OpenCV ، عکس مورد نظر را لود می کنیم.

```
# load image
image = cv2.imread('Camera Man.bmp')
# calculate histogram
pdf = calc_hitogram(image)
width , length , bands = image.shape

✓ 1.2s
```

سپس مطابق خط ۴، با استفاده از تابع ای که برای محاسبه هیستوگرام نوشته ام، هیستوگرام عکس را به‌دست می‌آوریم. تابع calc_histogram به شکل زیر تعریف می‌شود:

```
def calc_hitogram(image):

    length = image.shape[0]
    width = image.shape[1]

    pdf = np.zeros(256)

    for i in range(length):
        for j in range(width):
            k = image[i][j]
            pdf[k] +=1


    return pdf
```

در این تابع، عکس مورد نظر به عنوان ورودی تابع استفاده می‌شود. در دو حلقه تو در تو، فراوانی رخ داد هر پیکسل که مقداری بین ۰ تا ۲۵۵ دارد را، در یک آرایه به نام pdf ذخیره می کنیم. طول این آرایه ۲۵۶ است، و اندیس k این آرایه، نشان دهنده فراوانی پیکسل هایی با مقدار k است.

```
# plt.style.use('_mpl-gallery')
plt.style.use(plt.style.available[5])

# plot histogram
fig, ax = plt.subplots(figsize=(17,8))
x = np.arange(0,256,1)
y = pdf
ax.stem(x,y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 6501), yticks=np.arange(0, 6501,300))

plt.show()
```

✓ 0.2s

در گام آخر، مطابق قطعه کد بالا، با یک نمودار از نوع stem و با استفاده از کتابخانه matplotlib، نمودار هیستوگرام مربوط به عکس camera man را رسم می کنیم. در محور y این نمودار، مقادیر فراوانی یا همان pdf را جایگزاری می کنیم، و در محور x اعداد ۰ تا ۲۵۵.

نتیجه:

با اجرای این سلول ها، نمودار هیستوگرام عکس camera man را رسم می کنیم. در سه بخش این هیستوگرام تراکم پیکسل های را مشاهده می کنیم. بخش اول، از پیکسل هایی با مقدار ۲۰ تا اندکی پایین تر از ۱۰، که می‌توان دلیل آن را، وجود کت سیاه رنگ در تصویر باشد. همچنین یک رنج از پیکسل ها با مقادیر بین ۱۰۰ تا ۱۴۰ تا می‌توان به ناحیه ای که زمین را شامل می‌شود نسبت داد. رنج ۱۵۰ تا ۱۹۰ نیز می‌تواند نمایشگر بخش آسمان و فراوانی پیکسل های مربوط به این بخش باشد.

۲.۱.۱.۱

توضیح فنی:

در قطعه کدی مشابه تصویر زیر، مقدار هر پیکسل را بر ۳ تقسیم می‌کنیم. با اینکار انتظار داریم، تصویر تاریک‌تر شود. برای اینکار یک کپی از تصویر اصلی تهیه می‌کنیم و نام آن را D می‌گذاریم.

```
image = cv2.imread('Camera Man.bmp')

# decreasing brightness by dividing values by 3
D = np.copy(image)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        D[i][j] = image[i][j]/3

```

نتیجه:

در بخش بعد بررسی می‌شود.

۲.۱.۱.۲

توضیح فنی:

```
# plot original image histogram
fig, ax = plt.subplots(figsize=(17,8))
x = np.arange(0,256,1)
y = pdf
ax.stem(x,y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 6501), yticks=np.arange(0, 6501,300))

plt.show()

# plot D image histogram
D_pdf = calc_hitogram(D)
fig, ax = plt.subplots(figsize=(17,12))
x = np.arange(0,256,1)
y = D_pdf
ax.stem(x,y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 12001), yticks=np.arange(0, 12001,300))

plt.show()
```

✓ 1.9s

مطابق تصویر بالا، نمودار هیستوگرام عکس اصلی و عکس D را رسم می‌کنیم. هیستوگرام عکس D در خط ۱۳ محاسبه شده است. اجرای قطعه کد بالا، دو نمودار را به ترتیب ذکر شده رسم می‌کند.

نتیجه:

با اجرای سلول این بخش، در نمودار اول هیستوگرام عکس اصلی را رسم می‌کنیم که در بخش قبل نیز مشاهده شد. در نمودار دوم که مربوط به هیستوگرام عکس D است، تفاوت فاحشی بین فراوانی پیکسل ها با عکس اصلی داریم. فراوانی پیکسل های بخش های روشن تصویر، به طور معنی داری، کاهش یافته، و فراوانی پیکسل های بخش های تاریک‌تر تصویر، بسیار بیشتر شده است. به طور مثال، رنج پیک ۱۵۰ تا ۱۹۰ در عکس قبلی، به حدود ۵۰ تا ۶۵ مپ شده است و فراوانی این پیکسل ها بسیار زیادتر از عکس اول شده است.

۲.۱.۱.۳

توضیح فنی:

```
# first, we should normalize the pdf
normal_D_pdf = np.zeros(len(D_pdf))
normal_D_pdf = normalizeHistogram(D_pdf,D.shape[0],D.shape[1])
# calculating cdf
D_cdf = np.zeros(len(D_pdf))
D_cdf = calc_cdf(normal_D_pdf)
# re-map original image pixels to new ones
H = reMap(image,D_cdf)

plt.imshow(H)


✓ 0.5s
```

برای اجرای روال متعادل سازی هیستوگرام، در ابتدا نیاز داریم تا طبق الگوریتم ارائه شده در درس، هیستوگرام فعلی را نرمالایز کنیم. برای اینکار یک تابع به نام normalizeHistogram نوشته ایم.

```
def normalizeHistogram(pdf,width,length):
    normal_pdf = np.zeros(len(pdf))

    for i in range(len(normal_pdf)):
        normal_pdf[i] = pdf[i]/(width*length)

    return normal_pdf
```

در این تابع، مقادیر pdf را به رنج بین ۰ تا ۱ مپ می‌کنیم. چون ماکسیمم مقدار فراوانی برابر حاصل ضرب طول در عرض تصویر است، برای تقسیم کردن در مخرج از این عدد استفاده می‌کنیم.

در گام دوم، مطابق الگوریتم به محاسبه cdf (تابع چگالی) می‌پردازیم. برای اینکار تابع calc_cdf را تعریف می‌کنیم.

```python
def calc_cdf(normal_pdf):

    cdf = np.zeros(len(normal_pdf))

    for i in range(len(normal_pdf)):
        for j in range(i):
            cdf[i] += normal_pdf[j]

        cdf[i] *=255
        cdf[i] = round(cdf[i])

    return cdf
```

در این تابع، به ازای مقدا i ام از آرایه نرمال شده، تمام اندیس های ۰ تا i از آرایه نرمال شده را با هم جمع می‌کنیم. طبق فرمول محاسبه cdf، در حلقه بیرونی، که در واقع انتهای سیگمای جمع مقادیر نرمالایز شده تا آن اندیس است، مقدار آن اندیس را در ۲۵۵ ضرب می‌کنیم. در انتها چون این مقدار ممکن است یک عدد اعشاری باشد، آن را به مقدار صحیح نزدیک تر به آن گرد می‌کنیم.

در گام نهایی نیاز است تا با توجه به تابع تبدیل به دست آمده، مقادیر پیکسل های عکس را آپدیت کنیم. برای اینکار، از یک تابع به نام reMap نوشته ایم.

```python
def reMap(image,target_cdf):
    newImage = image.copy()
    for i in range(newImage.shape[0]):
        for j in range(newImage.shape[1]):
            newImage[i][j] = target_cdf[newImage[i][j]]

    return newImage

```

این تابع با دریافت عکس هدف و cdf مورد نظر، پیکسل های عکس را با مقادیر جدید بر اساس cdf داده شده، آپدیت می‌کند.

نتیجه:

اگر نتیجه این تبدیل را مشاهده کنیم، می‌بینیم که عکس دارای قسمت های بسیار سفید است، که به دلیل وجود شیب زیاد در تابع تبدیل به دلیل وجود قله هایی از فراوانی پیکسل، در هیستوگرام عکس ورودی است. یک روش حل این مشکل، برش زدن هیستوگرام است، که چون در صورت سوال رفع این مشکل خواسته نشده، رفع مشکل پیاده‌سازی نشده است. روش دیگر حل این مشکل، استفاده از متعادل سازی محلی است، که در بخش بعدی پیاده سازی شده است.

۲.۱.۱.۴

توضیح فنی:

```python
L = local_histo_equalization(D,32)
plt.imshow(L)
```

✓ 4.4s

برای پیاده‌سازی متعادل سازی محلی هیستوگرام، یک تابع پیاده سازی کرده ایم.

```python
def local_histo_equalization(image,windowSize):
    width , length , band =image.shape

    newImage = np.zeros_like(image)
    n = 0
    for i in range(0,length,windowSize):
        for j in range(0,width,windowSize):
            start = (j,i)
            end0 = j+windowSize
            end1 = i+windowSize

            if end0>width:
                end0 = width

            if end1>length:
                end1 = length

            end = (end0,end1)

            buffer = image[start[0]:end[0], start[1]:end[1]]

```

ورودی این تابع، یک عکس و ابعاد پنجره مورد نظر برای بخش‌بندی تصویر بر اساس این پنجره و محاسبات به صورت محلی در این پنجره است. مطابق شکل، ابتدا براساس اندازه پنجره، روی عکس حرکت می‌کنیم و یک برش با اندازه windowsize*windowsize از عکس را انتخاب می‌کنیم. متغییر بافر، یک متغییر میانی است که ابتدا محاسبات را برای هر برش از عکس، روی این تصویر انجام می‌دهیم، سپس نتایج را به متغییر مورد نظر برای عکس جدید منتقل می‌کنیم.

```python

            buffer_histo = calc_hitogram(buffer)
            normal_buffer = normalizeHistogram(buffer_histo,windowSize,windowSize)
            buffer_cdf = calc_cdf(normal_buffer)
            remaped_buffer = reMap(buffer,buffer_cdf)

            newImage[start[0]:end[0], start[1]:end[1]] = remaped_buffer

            n +=1

    eqaulizedImage = newImage
    return eqaulizedImage
```

در ادامه، مطابق حالت عادی متعادل سازی، برای هر بخش، روال متعادل سازی را روی برش انتخاب شده، اعمال می‌کنیم. در انتهای حلقه، بافر ریمپ شده را، در یک آرایه با نام newImage که درواقع عکس تولیدی جدید است، جایگزاری می‌کنیم.

نتیجه:

با مشاهده عکس تولیدی در این مرحله، می بینیم که عکس تولیدی در بخشی نقاط حالت پنجره پنجره دارد. دلیل این اتفاق، این است که در پنجره هایی که از نقاط یکدست تصویر نمونه برداری شده است، فقط فراوانی برخی مقادیر خاص از رنج ۰ تا ۲۵۵ زیاد است، و فراوانی بقیه مقادیر نزدیک به صفر است، و وجود این پیک ها متعادل سازی را کمی با مشکل مواجه می‌کند. اما در بخش هایی که پنجره ها، از قسمت هایی نمونه برداری کرده‌اند که شامل جزیئات عکس است، این حالن پنجره پنجره شدن کمتر دیده می‌شود و تا حدود زیادی، عملکرد محلی سازی بهتر از متعادل سازی کلی است، چون باعث می‌شود تا کنتراست بهتر تغییر کند و سایه ها و لبه ها نیز مشخص تر هستند.

۲.۱.۱.۵

توضیح فنی:

هیستوگرام های H و L را محاسبه و رسم می کنیم. میله های آبی رنگ روی نمودار، نشان دهنده مقادیر هیستوگرام تصویر H و میله های قرمز مربوط به تصویر L است.

```
# plot H image histogram (blue)
fig, ax = plt.subplots(figsize=(15,8))
x = np.arange(0,256,1)
y = calc_hitogram(H)
ax.stem(x,y)

# plot L image histogram (red)
y = calc_hitogram(L)
ax.stem(x,y)
markerline ,stemlines ,baseline = ax.stem(x,y,linefmt='red',markerfmt='x')
markerline.set_color('red')

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 6001), yticks=np.arange(0, 6001,200))

plt.show()
```

✓ 2.8s

نتیجه:

مطابق نمودار های رسم شده، میبینیم که هیستوگرام تصویر L ، نسبت به تصویر H از پراکندگی بیشتری برخوردار است. مطابق انتظار، هیستوگرام تصویر H، به دلیل اینکه شامل دو ناحیه سفید و سیاه است، به همین شکل باشد، یعنی در رنج بالاتر از ۲۵۰ و رنج ۵۰ تا ۶۰ ، فراوانی مقادیر بسیار زیاد است و در بقیه مقادیر، فراوانی ها کم یا صفر است. برعکس در تصویر L ، نموداری داریم که تقریبا فراوانی پراکنده ای دارد، و نمی توان یک قله مشخص و برجسته را در آن مشخص کرد، علی‌رغم اینکه در برخی مقادیر، یک پرش و افزایش فراوانی، نسبت به مقادیر همسایه خود دارد، که قابل چشم پوشی است.

۲.۱.۱.۶

توضیح فنی:

در سلول اول این بخش، تبدیل لگاریتمی مطابق فرمول ارائه شده در اسلاید درس پیاده سازی شده است. در ادامه نیز تصویر حاصل و هیستوگرام آن رسم شده است.

```
# log transform
c = 255 / np.log(1 + np.max(image))
D_log = log_transform(D,c)
D_log = np.array(D_log, dtype = np.uint8)
```

پیاده‌سازی تابع log_transform به شکل زیر است:

```

def log_transform(image, c):
    s = c*np.log(1+image)
    return s
```

در سلول دوم، تبدیل معکوس لگاریتمی پیاده‌سازی شده است.

```
# inverse log transform
base = 2
c = 255.0 / (pow(base, np.max(image)) - 1)
D_iLog = inverse_log_transform(D,c,base)
D_iLog = np.array(D_iLog, dtype = np.uint8)
```

پیاده‌سازی inverse_log_transform:

```
def inverse_log_transform(image , c, base):
    s = c*(np.power(base,image)-1)
    return s
```

در سلول سوم، تبدیل نمایی پیاده‌سازی شده است.

```
# power-law transform
gamma = 1.5
# c = 255 / np.power(image,gamma)
c = 2
D_power = power_law_transform(D,c,gamma)
D_power = np.array(D_power, dtype = np.uint8)
```

پیاده‌سازی power_law_transform:

```
def power_law_transform(image, c, gamma):
    s = c*np.power(image,gamma)
    return s
```

نتیجه:

مقادیر پارامتر ها برای توابع، با کمک و مطالعه این لینک و این لینک انتخاب شده است.

۲.۱.۲

توضیح فنی:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *

cameraman = cv2.imread("Camera Man.bmp")
width , length ,band = cameraman.shape
cm_histogram = calc_hitogram(cameraman)
cm_histogram_normal = normalizeHistogram(cm_histogram,width,length)
cm_cdf = calc_cdf(cm_histogram_normal)
newCameraMan = reMap(cameraman,cm_cdf)
ncm_histogram = calc_hitogram(newCameraMan)

```

مطابق تصویر و روالی که در قسمت ۲.۱.۱.۳ نیز توضیح داده شده، عکس camera man را لود کردیم و مراحل متعادل سازی را روی آن اجرا کرده ایم. در ابتدا هیستوگرام عکس با تابع calc_histogram محاسبه شده، سپس با تابع normalizeHistogram، هیستوگرام به‌دست آمده را نرمالایز کرده ایم. در گام بعد، cdf را با تابع calc_cdf محاسبه کرده ایم. در نهایت نیز بر اساس cdf محاسبه شده، عکس جدید camera man را ساخته ایم. در گام نهایی نیز هیستوگرام عکس جدید را حساب کرده ایم.

```

figure = plt.figure(figsize=(10,10))

figure.add_subplot(1,2,1)
plt.imshow(newCameraMan)
plt.title("equalized",color='black')

figure.add_subplot(1,2,2)
plt.imshow(cameraman)
plt.title("original",color='black')

```

قطعه کد نمایش عکس اصلی و عکس متعادل شده.

```
# plot original image histogram
fig, ax = plt.subplots(figsize=(15,5))
x = np.arange(0,256,1)
y = cm_histogram
ax.stem(x,y)

y = ncm_histogram

markerline ,stemlines ,baseline = ax.stem(x,y,linefmt='red',markerfmt='x')
markerline.set_color('red')

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 6601), yticks=np.arange(0, 6601,200))

plt.show()
```

قطعه کد رسم نمودار هیستوگرام عکس اصلی و عکس جدید.

نتیجه:

مطابق انتظار، کنتراست تصویر بهبود یافته است، پراکندگی فراوانی در نمودار متعادل شده بیشتر است و تفاوت بین لبه های سیاه و سفید واضح تر شده است.

۲.۲.۱

توضیح فنی:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *

he1 = cv2.imread('HE1.jpg')
he2 = cv2.imread('HE2.jpg')
he3 = cv2.imread("HE3.jpg")
he4 = cv2.imread('HE4.jpg')

```

قطعه کد لود کردن تصاویر.

```
local_eq1_he1 = local_histo_equalization(he1,16)
local_eq2_he1 = local_histo_equalization(he1,32)
local_eq3_he1 = local_histo_equalization(he1,128)

local_eq1_he2 = local_histo_equalization(he2,16)
local_eq2_he2 = local_histo_equalization(he2,32)
local_eq3_he2 = local_histo_equalization(he2,128)


local_eq1_he3 = local_histo_equalization(he3,16)
local_eq2_he3 = local_histo_equalization(he3,32)
local_eq3_he3 = local_histo_equalization(he3,128)

local_eq1_he4 = local_histo_equalization(he4,16)
local_eq2_he4 = local_histo_equalization(he4,32)
local_eq3_he4 = local_histo_equalization(he4,128)

```

قطعه کد پردازش تصاویر به صورت محلی، به ترتیب با پنجره هایی با سایز ۱۶،۳۲ و ۱۲۸.

در ادامه قطعه کد این سلول نیز تصاویر متعادل شده، نمایش داده شده اند.

نتیجه:

در بحث تاثیر ابعاد پنجره، در پنجره های بسیار کوچک، عکس یک حالت پنجره-پنجره شدن پیدا می‌کند، و چون هر پنجره به مقادیر ۰ تا ۲۵۵ اسکیل می‌شود، در هر پنجره کوچک، نقاط بسیار سفید و بسیار تیره پدید می‌آید، که در حالی که برای تصاویر شلوغ، شاید باعث کنتراست بهتر شود، اما قطعا برای نواحی یکدست(فرکانس تغییرات کم) باعث پدیدار شدن حالت نویزی می‌شود. در سمت مقابل، پنجره بزرگتر، شاید اثر نویزی کمتری داشته باشد، اما در نواحی که تغییرات بسیار شدید است، نسبت به پنجره با ابعاد وچکتر، کنتراست کمتری ایجاد می کند. بنابر توضیح بالا، باید با توجه به هر تصویر، تصمیم گرفت که از چه سایز پنجره ای برای متعادل سازی محلی استفاده کرد. با توجه به تصاویر داده شده، سایز پنجره ۱۲۸ یا ۶۴ حالت مناسبی است، که هم کنتراست خوبی ایجاد می کند، و هم عکس حالت نویزی پیدا نمی کند.

۲.۱.۳

توضیح فنی:

تابع histeq ، با کمک متعادل‌سازی هیستوگرام، شدت روشنایی پیکسل های تصویر را تغییر می‌دهد، در حالی که تابع imadjust ، شدت روشنایی پیکسل ها را، فارغ از متعادل‌سازی هیستوگرام انجام می‌دهد.

نتیجه:

ندارد